Big-Endian HP-UX and Little-Endian NT on HP's IA-64 Based Systems

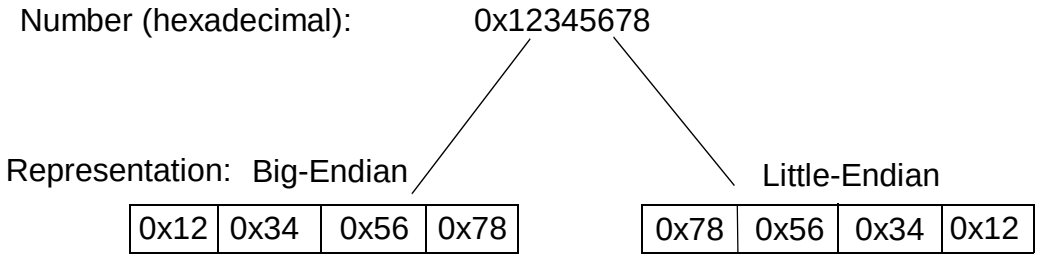
A brief look at HP-UX and NT having different endian representation for numbers on IA-64. The conclusion is that this is not an issue on which to base your machine selection.

1.0 The Endian Issue

The IA-64 architecture allows an operating system to choose between the two common representations of numbers: little-endian machines store the low-order byte of an integer first, while big-endian have the high-order byte first. Figure 1 shows both representations. For the most part, programs are independent of the choice of endian-ness. Some programs can become dependent on this choice. The major consequence, however, is the way numbers are represented on storage (disks, tape, etc.).

Computer centers often contain machines having different endian conventions. They can communicate over networks or share files because of well-established interchange standards used by programs such as databases, word processors, and network protocol stacks. These programs utilize standards, or create their own with full knowledge of the problem. Programs written for a single type of machine often ignore this issue, have no

Figure 1 Big-Endian and Little-Endian integer representations



such standard, and work fine within their own world. But if they were moved to an environment having the opposite convention, old data files would require conversion.

The existence of the two conventions is primarily due to history. Big-endian machines generally come from a history of larger machines, the mainframes or systems styled after mainframes. The little-endian machines are generally derived from 8-bit systems. Most models of computers are big-endian, but the large number of personal computers make little-endian the most widely used computers.

Bi-endian machines were invented to allow hardware to run multiple operating systems which have adopted different endian conventions. These include MIPS, Power-PC and PA-RISC.

The IA-64 architecture is bi-endian: it can accommodate numbers in either format. A given operating system makes a choice, and runs programs in that mode. NT, being a little-endian operating system, will continue to run in little-endian mode on IA-64. HP-UX, being a big-endian operating system, will continue to run in big-endian mode. Communication between the systems will take place as it does today.

2.0 Coping with Different Endian Conventions

The problem of coping with different endian numbers has been resolved. Had it not been, today's reality of heterogeneous networks, or client-server systems of heterogeneous machines would never have happened. When computers communicate, they use standard protocols which have selected a particular representation of numbers. Machines which use a different representation translate between the particular standard

Figure 2 Endian Conventions of Selected Operating Systems

OS	Hardware	Endian
NT	IA-32, DEC alpha, MIPS, IA-64	Little-Endian
Mac OS	Power-PC	Big-Endian
HP-UX	PA-RISC, IA-64	Big-Endian
Solaris	Sparc	Big-Endian
Solaris	IA-32, IA-64	Little-Endian
DEC Unix	Alpha	Little-Endian
AIX	IBM, Power-PC	Big-Endian
Irix	MIPS	Big-Endian
SCO Unix	IA-32, IA-64	Little-Endian

and the machine's internal representation. This translation is trivial, and done with virtually no performance impact.

The most common way to communicate between different machines uses character formats. Even here, character conversions are required between different standards. But high-speed communication with databases and other applications requires that binary data also be exchanged. Binary data must be converted between the representation on the database server and that of the client.

Conversion for databases normally takes place in the communication protocols. A standard format is normally provided, with conversion to and from that format. Some systems provide a communication format which indicates the format used for transmission; the receiver then converts if necessary.

Some applications support a common binary file format. FrameMaker is an example: it stores information in binary files which may be used on NT, Unix, and Macintosh systems. This compatibility has been in place for many years. It is established, known technology.

Endian differences may even be hidden when direct memory addressing is involved. For many years now, object database companies have delivered products where the data is compatible with multiple processors, even though the data is mapped into virtual memory. Programs on either NT or HP-UX may address objects in a shared object database, and the endian issues are resolved automatically by the Object-Oriented Database system. This conversion happens as the data is received and prepared for mapping into memory.

There is a concern, of course. An application could produce a binary file without any provision for retaining data format information. Such applications exist today, typically running in a single operating environment. Should such a program need to move to a different operating system having different endian convention, a few pieces of the program logic may need to be modified. This amount of change is usually much less than modifications needed to port to a different operating system. A more significant problem is enabling the program to read existing data files. The files need to have fields converted to the different endian convention. If the program can recognize an old file, a modified input routine can be created. Otherwise, the file will need to be converted by an external process. The complexity of this task will depend on how simple the file format is. If the format is undocumented, conversion can be difficult.

New application developers might be concerned that they not create a future data compatibility problem by developing for one specific endian convention. Their best recourse is to follow the coding practices for protecting their application from other sources of change. File and communication formats should always be identified with a version identification, and the data should be accessed through a version adapter. Data recorded with one endian convention should always be distinguished by a different version label or type label from data recorded with the other convention. In this way the mere addition of an adapter will allow the conversion of numbers. Files should not be created which have binary data but no record of its structure.

3.0 Enter IA-64

The IA-64 architecture is bi-endian, so it can support either endian convention. This enables many existing operating systems to port to IA-64 without changing endian conventions. There are promised implementations of NT, HP-UX, Solaris, and DEC Unix on this architecture. Most operating systems will choose to maintain the same endian convention that they've used historically, so that customers won't have the effort of converting files or applications. Communication between systems that have made different choices will continue to work as it does today.

There is nothing new here; it's the same for existing architectures that have bi-endian support.

4.0 Conclusion: No Problem!

There are many considerations which must be made in selecting a new computer system. Many factors influence the future utility of the system for solving enterprise problems with low cost of ownership. Near the bottom of the list is the endian conventions of the system. This is a solved problem which is of little relevance to computer system selection.